Structured Variable Elimination avec analyse en d-séparation

Lionel Torti, Pierre-Henri Wuillemin

Structured Variable Elimination avec analyse en d-séparation

Lionel Torti* — **Pierre-Henri Wuillemin***

avenue du Président Kennedy, 75016 Paris
lionel.torti@lip6.fr
pierre-henri.wuillemin@lip6.fr

RÉSUMÉ. *Au cours des dix dernières années, de nouveaux modèles basés sur les réseaux bayésiens sont apparus pour modéliser des systèmes complexes de grandes tailles. Ces nouveaux modèles appartiennent à deux familles : la première est issue de l'unification de la logique du premier ordre et des probabilités (Markov Logic Networks, Bayesian Logic) ; la deuxième d'une extension des réseaux bayésiens (Probabilistic Relational Models, Multy-Entity Bayesian Networks, Relational Bayesian Networks). Dans cet article, nous proposons une amélioration de Structured Variable Elimination (SVE), l'unique algorithme d'inférence dans les PRM, reposant sur une technique classique d'élagage des réseaux bayésiens : l'analyse en d-séparation. Nous montrons comment intégrer une telle analyse dans SVE et l'amélioration considérable qu'elle apporte pour l'inférence.*

ABSTRACT. *In the last ten years, new models based on Bayesian Networks emerged to handle large and complex systems. These new models can be divided in two: the unification with First Order Logic and uncertainty (Markov Logic Networks, Bayesian Logic) and Knowledge Base Construction Models (Probabilistic Relational Models, Multy-Entity Bayesian Networks, Relational Bayesian Networks). Structured Variable Elimination (SVE) is the state-of-the-art algorithm for Probabilistic Relational Models. In this paper, we propose an enhancement of SVE based on a well known complexity reduction technique from Bayesian Networks. We show how to integrate a d-separation analysis in SVE and how this leads to important improvements for the inference task.*

MOTS-CLÉS : *Modèles graphiques, inférence, réseaux bayésiens, Probabilistic Relational Models, d-séparation*

KEYWORDS: *Graphical models, inference, Bayesian Networks, Probabilistic Relational Models, d-separation*

1. Introduction

Les réseaux bayésiens (RB) sont un formalisme largement utilisé pour le raisonnement dans l'incertain. Présentés il y a vingt ans (Pearl, 1988), ils sont encore utilisés pour un grand nombre d'applications dans des domaines tels que le diagnostique automatique, la fiabilité, la gestion des risques ou encore la sûreté de fonctionnement (SKOOB, 2007). Le succès des RB dans les applications industrielles crée un problème de passage à l'échelle dans la complexité et la taille des réseaux. En effet, les RB possèdent plusieurs défauts qui rendent leurs créations, leurs maintenances et leurs exploitations difficiles. De nouveaux formalismes, communément appelés *First Order Probabilistic Models* (FOPM), fournissent différentes propositions pour combler les lacunes des RB. Ces nouveaux modèles peuvent être répartis en deux familles. La première regroupe les propositions d'unification de la logique du premier ordre et des probabilités : *Markov Logic Networks* (Domingos *et al.*, 2007), *Bayesian Logic* (Kerstring *et al.*, 2000), etc. La deuxième regroupe les extensions des réseaux bayésiens : *Probabilistic Relational Models* (Pfeffer, 2000), *Multy-Entity Bayesian Networks* (Laskey, 2008), *Relational Bayesian Networks* (Jaeger, 1997), etc.

Après avoir introduit les concepts princiaux des PRMs, nous présentons *Structured Variable Elimination* avec analyse en d-séparation (SVED), une amélioration de l'algorithme *Structured Variable Elimination* (Pfeffer, 2000) dans lequel nous avons inclus une analyse en d-séparation du PRM. Alors que *Structured Variable Elimination* (SVE) exploite l'information structurelle encodée dans un PRM, la d-séparation contribue à réduire la quantité nécessaire d'information requise pour répondre à une requête. Toutefois, les deux méthodes peuvent sembler incompatibles puisque l'analyse en d-séparation *casse* l'information structurelle. Nous verrons qu'un léger changement dans le formalisme des PRM nous permet d'utiliser conjointement les deux méthodes. De plus, nous montrerons comment l'algorithme BayesBall (Shachter, 1998) peut être aisément adapté aux PRM.

2. Probabilistic Relational Models

Les PRM sont une amélioration des réseaux bayésiens orientés objets (Koller *et al.*, 1997) et (Bangsö *et al.*, 2000). À l'instar de la programmation orientée objet, l'élément principal d'un PRM est la classe. Une classe définit une famille d'objets partageant des propriétés communes : graphe, attributs et références. Le graphe et les attributs définissent un fragment de réseau bayésien. Les références et les chaînes de références permettent de créer des dépendances entre les attributs de classes différentes. Pour des raisons de concision, nous donnons ici une présentation succincte des PRM. Pour une présentation plus détaillée, nous suggérons (Pfeffer, 2000) et (Getoor *et al.*, 2007).

Définition 1. Une classe X est définie par un graphe orienté sans circuit G_X sur un ensemble d'attributs, de références et de chaînes de références (respectivement noté $\mathcal{A}(X)$, $\mathcal{R}(X)$ et $\mathcal{C}(X)$).

Nous notons par $\mathcal{X}$ l'ensemble des classes d'un PRM. La définition 1 fait appel à des concepts (attribut, référence, ...) que nous nous attachons maintenant à définir.

Définition 2. Un attribut de la classe X, noté $A \in \mathcal{A}(X)$, est représenté par un nœud dans G_X. Un attribut est associé à une variable aléatoire et à une fonction de probabilité conditionnelle définie sur A et ses parents $\Pi(A)$ dans G_X.

Un attribut est l'équivalent d'un nœud dans un RB. Le terme «fonction de probabilités conditionnelles» est préféré à celui de «table de probabilités conditionnelles» en raison de la présence d'agrégateurs. Avant de pouvoir définir la notion d'agrégateurs, nous devons d'abord définir les notions de référence et de chaîne de références.

Définition 3. Une référence de la classe X, notée $\rho \in \mathcal{R}(X)$, définit une relation entre deux classes et peut être simple (une relation un à un) ou complexe (une relation n à n). Le domaine d'une référence est sa classe, noté $Dom[\rho] = X$, tandis que sa portée est la classe qu'elle référence, notée $Range[\rho] = Y$ ($Y \in \mathcal{X}$).

Définition 4. Une chaîne de références $\rho_1.\rho_2 \ldots \rho_n.A$ est une liste de références telle que $\forall i < n, Range[k] = Dom[k+1]$. Elle est terminée par un attribut tel que $A \in \mathcal{A}(Range[\rho_n])$. Une chaîne de références est simple si toutes ses références sont simples, sinon elle est complexe.

Une chaîne de références est donc la représentation d'un attribut externe à une classe dans le contexte de cette classe ; ce qui permet à un attribut interne de déclarer cet attribut externe comme un parent.

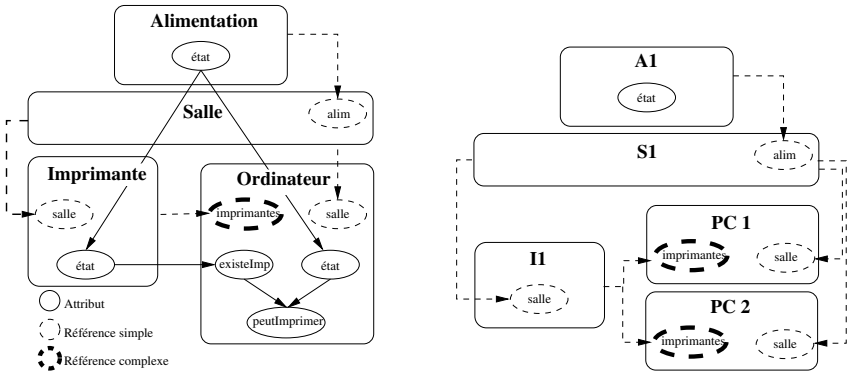
Par analogie avec la programmation orientée objet, nous utilisons des notations pointées pour désigner un membre d'une classe. Soient une classe X et un attribut A de X, nous écrirons indifféremment $X.A$, $A \in \mathcal{A}(X)$ ou encore directement A en l'absence d'ambiguïté. Les mêmes notations s'appliquent pour une référence ou une chaîne de références.

La figure 1a est une illustration des notions précédentes. Elle modélise un problème de diagnostic : étant donné un parc informatique constitué de plusieurs ordinateurs et d'imprimantes répartis dans différentes salles, nous souhaitons savoir s'il est possible d'imprimer depuis un ordinateur donné. Un ordinateur ou une imprimante se trouve dans une salle ; ce qui est représenté par la référence simple `salle` dans `Imprimante` et `Ordinateur`. Un ordinateur peut être connecté à une ou plusieurs imprimantes. Ceci est représenté par la référence complexe `imprimantes` dans la classe `Ordinateur`. La classe `Salle` est notable puisqu'elle ne sert qu'à donner accès à la classe `Alimentation` pour les machines qui référencent cette `Salle`. Dans les quatre classes de la figure 1a, il existe trois chaînes de références :

– le parent de `Imprimante.état` : `salle.alim.état` ;

– le parent de `Ordinateur.état` : `salle.alim.état` ;

– le parent de `Ordinateur.existeImp` : `imprimantes.état`.



(a) Un exemple de classes. (b) Un exemple de système.

Figure 1. *Un exemple de graphe de dépendances des classes et d'un système.*

Un cas particulier est l'attribut `existeImp` de la classe `Ordinateur`. Cet attribut possède un parent obtenu par l'utilisation d'une chaîne de références complexe et il peut donc posséder un nombre quelconque de parents : ce qui permet de spécifier le branchement d'un ordinateur à une ou plusieurs imprimantes. Ceci pose problème pour définir sa table de probabilités conditionelles (TPC), puisque celle-ci nécessiterait de connaître le nombre exact de parents de l'attribut pour être définie. Pour résoudre ce problème, nous utilisons des fonctions déterministes permettant de générer automatiquement la TPC pour n'importe quel nombre de parents. En spécifiant une famille de fonctions autorisées (*exists*, *for all*, *mean*, *min*, *max*, ...) il est possible d'agréger l'information représentée par une chaîne de références complexe. Un attribut dont la TPC est ainsi définie est appelé un agrégateur. Nous terminons cette section par les définitions d'un système, d'instances d'une classe et du RB sous-jacent d'un système.

Définition 5. Une instance est l'utilisation d'une classe X dans un système σ_s. Il peut y avoir plusieurs instances d'une même classe.

Nous notons par $\mathcal{I}(X)$ l'ensemble des instances de X. L'ensemble des notations que nous utilisons pour une classe peuvent être appliquées à une instance : tout comme sa classe, une instance possède un graphe, des attributs, des références et des chaînes de références. Les références d'une instance possèdent également un domaine et une portée, à ceci près qu'elles réfèrent des instances et non plus des classes. En particulier, la portée d'une référence instanciée renvoie un ensemble d'instances. Dans le cas d'une référence simple cet ensemble est un singleton.

Définition 6. Un système (ou squelette relationnel) est un ensemble d'instances $\mathcal{I}$ tel que chaque référence d'une instance $x \in \mathcal{I}$ soit correctement définie, i.e. $\forall X \in \mathcal{X}, \forall x \in \mathcal{I}(X), \forall \rho \in \mathcal{R}(x), \exists y \in \mathcal{I}(Range[X.\rho])$ et $Range[x.\rho] = y$.

La figure 1b représente un système composé d'une alimentation (A1), une salle (S1) contenant une imprimante (I1) reliée à 2 ordinateurs (PC1 et pC2). Nous avons précisé que les PRM sont une extension des RB, à ce titre il est toujours possible de construire un RB équivalent à un système. Un tel RB est appelé réseau bayésien sous-jacent d'un système. Il est construit en créant un nœud par attribut et en ajoutant des arcs entre les nœuds de manière à respecter les dépendances existantes dans le système. Un algorithme de génération de RB sous-jacent linéaire en temps peut être trouvé dans (Pfeffer, 2000).

Nous avons présenté très succinctement les PRM qui permettent de définir univoquement des modèles probabilistes de grande taille. L'inférence dans un tel modèle peut utiliser la structure du PRM afin d'être plus efficace. C'est le principe de l'algorithme SVE que nous explicitons maintenant.

3. *Structured Variable Elimination*

Structured Variable Elimination (SVE) (Pfeffer, 2000) est une extension de l'algorithme d'inférence *Variable Elimination* (VE) (Dechter, 1999). SVE exploite l'information structurelle d'un PRM pour empêcher la répétition de certains calculs. Afin d'exploiter l'information structurelle, nous devons différencier deux types d'attributs : les attributs internes et externes. Les attributs externes sont ceux qui possèdent des enfants définis dans une autre classe. Dans la figure 2, A, B, C, D et E sont des attributs internes ; F et G sont des attributs externes.

La première phase de SVE consiste à proposer une élimination des nœuds internes ; élimination qui s'effectue au niveau de la classe et non dans le système. Cette nuance implique que cette opération est effectuée une unique fois, quelque soit le nombre d'instances de cette même classe. Le tableau 1 montre les potentiels obtenus après l'élimination des attributs internes.

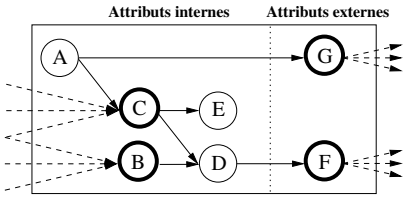


Figure 2. *Une classe est ses attributs internes et externes.*

Une fois tous les attributs internes éliminés, les potentiels obtenus contiennent uniquement des attributs externes : ceux de la classe et ceux définis dans d'autres classes et dont dépendaient certains attributs de la classe. SVE exploite ce résultat en éliminant les attributs internes au niveau des classes et en réutilisant les potentiels obtenus pour chaque instance. Il existe toutefois une catégorie d'attributs ne pouvant être éliminée de cette manière : les agrégateurs. La table de probabilités conditionnelles (TPC) d'un

Attributs éliminés	Potentiels
$\{\ \}$	$\{\ P(A), P(G\|A), P(B\|\Pi(B)),$ $P(C\|A,\Pi(C)), P(D\|B,C),$ $P(E\|C), P(F\|D)\ \}$
$\{A\}$	$\{\ \Phi_1(C,G,\Pi(C)), P(B\|\Pi(B)),$ $P(D\|B,C), P(E\|C), P(F\|D)\ \}$
$\{A,B\}$	$\{\ \Phi_1(C,G,\Pi(C)), P(E\|C),$ $\Phi_2(C,D,\Pi(B)), P(F\|D)\ \}$
$\{A,B,C\}$	$\{\ \Phi_3(D,E,G,\Pi(B),\Pi(C)),$ $P(F\|D)\ \}$
$\{A,B,C,D\}$	$\{\ \Phi_4(E,F,G,\Pi(B),\Pi(C))\ \}$
$\{A,B,C,D,E\}$	$\{\ \Phi_5(F,G,\Pi(B),\Pi(C))\ \}$

Tableau 1. *Élimination des attributs internes de la figure 2.*

agrégateur est automatiquement générée en fonction du contexte de l'instance (i.e. elle est fonction du nombre de parents de l'agrégateur). Ceci ne permet pas de raisonner au niveau des classes pour ces attributs. Finalement, s'il existe des observations sur un ou plusieurs attributs internes d'une instance, il n'est plus possible d'utiliser l'élimination réalisée au niveau des classes. Il est en effet nécessaire d'intégrer les observations dans le calcul de l'élimination.

Si nous appliquons une élimination des attributs internes à toutes les instances d'un système, il en résulte un ensemble de potentiels contenant uniquement des attributs externes et des agrégateurs. Pour éviter une inférence en deux temps (élimination des attributs internes, puis des externes), SVE utilise un ordre d'élimination de *bas en haut* pour éliminer récursivement les instances : une instance ne peut être éliminée que quand toutes les instances utilisant ses noeuds externes ont été éliminées. Tout comme dans VE, l'algorithme se termine une fois que toutes les instances sont éliminées.

4. Analyse en d-séparation dans un PRM

Telle qu'elle est indiquée dans (Shachter, 1998), l'analyse en d-séparation (vis à vis de la structure et des observations) peut induire une autre forme d'optimisation de l'inférence : elle permet d'élaguer le RB pour ne prendre en compte que l'ensemble minimal de variables aléatoires pour le calcul d'une marginale.

L'algorithme BayesBall (Shachter, 1998) exploite la d-séparation pour déterminer via une méthode simple l'ensemble des variables aléatoires requises vis à vis d'une requête et d'un ensemble d'observations. L'algorithme utilise une balle imaginaire qui rebondit entre les nœuds d'un RB de manière à suivre le flux d'informations. Les travaux présentés dans cet article ont pour but d'adapter cette méthode aux PRM. Une analyse en d-séparation d'un PRM doit permettre de réduire le nombre de calculs nécessaires et peut être perçue comme l'utilisation de l'information structurelle de bas niveau, en opposition à l'information de haut niveau, représentée par les classes et le squelette relationnel.

Il existe deux approches possibles pour exploiter l'algorithme BayesBall avec les PRM. Une approche naïve serait de réaliser l'analyse sur un RB sous-jacent et d'utiliser une bijection entre les nœuds du RB et les attributs du système pour connaître l'ensemble d'attributs requis. Toutefois, SVE n'utilisant pas les RB sous-jacents, il serait contraignant de devoir générer le RB puisque cela induit un coût en mémoire élevé. L'autre approche consiste à adapter l'algorithme BayesBall aux PRM. Ceci peut être réalisé grâce à une modification intuitive de la représentation des classes et des instances. La représentation classique pose problème car elle ne contient pas l'information des parents et des enfants d'un attribut (i.e. le voisinage d'une classe et par extension d'une instance). Nous avons vu que les PRM utilisent des chaînes de références pour définir des dépendances entres attributs de différentes classes. Toutefois, chaque chaîne de références possède une inverse. Une chaîne de références inverse est le chemin inverse représenté par une chaîne de références.

Par exemple, si nous nous penchons sur la figure 1a et en particulier l'attribut `Imprimante.état`. Cet attribut possède un enfant, l'attribut `Ordinateur.existeImp`, qui accède à `Imprimante.état` grâce à la chaîne de références `imprimantes.état`. Cette chaîne de références possède une inverse constituée d'inverses des références y appartenant : `inv(imprimantes).existeImp`. En ajoutant un nœud pour chaque chaîne de références et inverse de chaîne de références, ainsi que des arcs représentant les dépendances probabilistes, il est possible d'utiliser l'algorithme BayesBall si nous considérons ces nœuds comme des attributs sans observation. En faisant rebondir la balle d'une instance à une autre lorsqu'elle arrive sur une chaîne de références, déterminer l'ensemble d'attributs requis devient trivial.

5. *Structured Variable Elimination* avec analyse en d-séparation

L'algorithme 1 (SVED) décrit le fonctionnement de SVE avec d-séparation. Ses arguments sont une instance (Q), un attribut (att) de Q dont nous voulons la marginale et le dernier argument est un ensemble vide de potentiels qui contiendra la marginale de l'attribut att après l'appel de SVED. Les deux procédures 2 (ElDesc) et 3 (ElAsc) sont utilisées pour éliminer récursivement les instances filles et les instances parentes d'une instance.

À la ligne 1 de SVED, les deux ensembles *visité* et *non_visité* servent respectivement à prévenir plusieurs éliminations d'une même instance et à retarder l'élimination d'autres. Il est nécessaire de reporter l'élimination de certaines instances pour respecter l'ordre de *bas en haut* imposé par SVE. En effet, quand une instance i est éliminée, l'ordre d'élimination utilisé garantit que ElDesc a été appelé sur chaque instance fille de i. Toutefois, les instances parentes des enfants de i peuvent ne pas être encore éliminées. Il faut donc en garder la trace pour les éliminer au moment opportun (ligne treize de la procédure ElDesc). Ces instances sont placées dans l'ensemble *non_visité* et sont éliminées si l'instance courante les précède (ligne quatorze de SVED et ligne quinze de ElAsc).

Algorithme 1 : SVED

```
   Entrées : Q, att, saut
1  Ensemble visitée, non_visitée;
2  visitée = {Q} ∪ visitée;
3  saut = Requis(Q) ∪ saut;
4  pour n ∈ Requis(Q) faire
5      si n est une chaîne de références inverse alors
6          pour i ∈ Instance(n) faire
7              ElDesc(i, saut, visitée, non_visitée);
8  VE(Q, att, Requis(Q), saut);
9  pour i ∈ non_visitée faire si Q précèdes i alors
10     ElDesc(i, saut, visitée, non_visitée);
11 pour n ∈ Requis(Q) faire
12     si n est une chaîne de références alors
13         pour i ∈ Instance(n) faire
14             ElAsc(i, saut, visitée, non_visitée);
```

Pour savoir si une instance peut appeler la procédure ElDesc sur un de ses enfants, nous utilisons une relation d'ordre sur les classes. Pour deux classes C_1 et C_2, s'il existe un couple d'attributs (A, B), avec $A \in \mathcal{A}(C_1)$ et $B \in \mathcal{A}(C_2)$, de sorte que A soit un ancêtre de B, alors C_1 précède C_2. Si C_1 précède C_2 et C_2 précède C_1 alors il est nécessaire de choisir arbitrairement l'ordre d'élimination des deux instances. Dans les deux cas, les lignes trois de SVED, ElDesc et ElAsc garantissent la viabilité des calculs. La sous-procédure VE correspond à un appel à l'algorithme *Variable Elimination* (Dechter, 1999) sur l'ensemble des nœuds requis pour l'instance courante (ligne huit de SVED et ligne neuf de ElDesc et ElAsc). Lorsque VE reçoit une classe comme argument, il réutilise n'importe quel calcul précédemment réalisé (ligne neuf de ElDesc et ElAsc). La fonction *Instance* renvoie l'ensemble des instances assignées à une chaîne de références ou chaîne de références inverse.

6. Résultats expérimentaux

Les expériences présentées dans cette section utilisent des systèmes générés à partir du graphe de dépendance des classes de la figure 1a et en utilisant l'implémentation des PRM présente dans la librairie aGrUM (aGrUM, 2008) développée au LIP6. Les exemples que nous avons utilisés sont généralement composés d'un nombre fixe de pièces et un nombre variable d'imprimantes et d'ordinateurs. Le nombre d'ordinateurs influence la taille du système, tandis que le nombre d'imprimantes a un impact direct sur la taille de la plus grosse clique. Ce deuxième point est dû à la présence d'un agrégateur dans chaque instance de la classe `Ordinateur` (l'attribut `existeImp`).

Procédure `Élimination descendante(`*ElDesc*`)`

Entrées : i, $saut$, $visitée$, $non\_visitée$
Entrées : i, $saut$, $visitée$, $non\_visitée$
si $i \notin visitée$ **alors**
  $visitée = \{i\} \cup visitée$;
  $saut = Requis(i) \cup saut$;
  pour $n \in Requis(i)$ **faire**
    si n *est une chaîne de références inverse* **alors**
      pour $j \in Instance(n)$ **faire**
        ElDesc(i, $saut$, $visitée$, $non\_visitée$);
  si $estObservée(i)$ **alors** VE(i, $Requis(i)$, $saut$);
  sinon VE($Classe(i)$, $Requis(Classe(i))$, $saut$);
  pour $n \in Requis(i)$ **faire**
    si n *est une chaîne de références* **alors**
      pour $j \in Instance(n)$ **faire**
        $non\_visitée = \{j\} \cup non\_visitée$;

Procédure `Élimination ascendante(`*ElAsc*`)`

Entrées : i, $saut$, $visitée$, $non\_visitée$
si $i \notin visitée$ **alors**
  $visitée = \{i\} \cup visitée$;
  $saut = Requis(i) \cup saut$;
  pour $n \in Requis(i)$ **faire**
    si n *est une chaîne de références inverse* **alors**
      pour $j \in Instance(n)$ **faire**
        ElDesc(j, $saut$, $visitée$, $non\_visitée$);
  si $estObservée(i)$ **alors** VE(i, $Requis(i)$, $saut$);
  sinon VE($Classe(i)$, $Requis(Classe(i))$, $saut$);
  pour $j \in non\_visitée$ **faire** **si** i *précèdes* j **alors**
    ElDesc(j, $saut$, $visitée$, $non\_visitée$);
  pour $n \in Requis(Q)$ **faire**
    si n *est une chaîne de références* **alors**
      pour $i \in Instance(n)$ **faire**
        ElAsc(i, $saut$, $visitée$, $non\_visitée$);

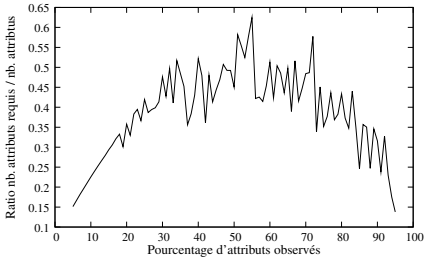
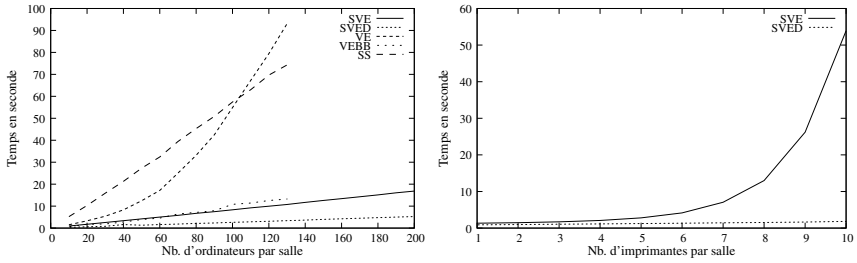


Figure 3. *Analyse en d-séparation avec BayesBall.*

La figure 3 illustre une expérience où nous avons observé aléatoirement un pourcentage croissant d'attributs dans un système composé de cinquante pièces, dix imprimantes et quarante ordinateurs (pour un total de $6\,501$ nœuds). L'attribut de la requête est choisi aléatoirement. La courbe de la figure 3 montre à quel point l'analyse en d-séparation est sensible à l'ensemble d'observations et à la requête. La couverture de markov d'un attribut le d-sépare des autres attributs du système lorsque celle-ci est observée, ce qui explique que la courbe tend vers un ratio de zéro lorsque le nombre d'observations augmente. De même, lorsque le pourcentage d'observations est faible, seuls les ancêtres sont nécessaires pour le calcul de la marginale d'un attribut. Ce test est fait sur un unique PRM de grande taille. Malheureusement, générer des classes et des systèmes n'est pas trivial et nous collaborons avec des industriels pour créer des cas réels d'utilisation des PRM, collaboration qui n'a pas encore porté ses fruits. Dans l'attente d'obtenir des systèmes plus stressants pour l'inférence dans les PRM, cet exemple nous permet toutefois d'espérer de bonnes performances.



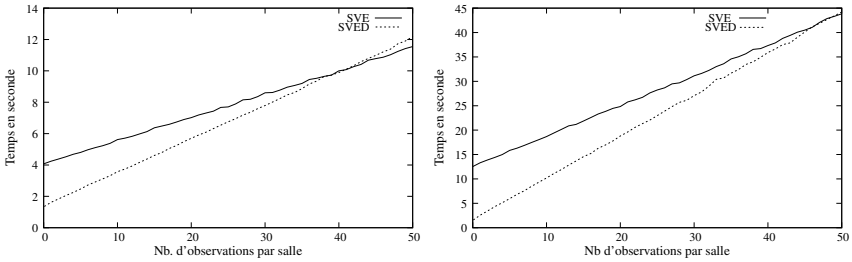
(a) Limite des inférences sous-jacentes. (b) Comparaison entre SVE et SVED.

Figure 4. *Performances d'algorithmes d'inférence en l'absence d'observations.*

La figure 4a illustre le comportement de différents algorithmes d'inférence en l'absence d'observation. Pour cette expérience nous avons augmenté le nombre d'ordinateurs par salle de dix à deux cents, avec six imprimantes et quarante salles (pour un total de $1\,441$ à $24\,241$ nœuds). Nous pouvons constater que la courbe s'arrête après cent trente ordinateurs pour les algorithmes utilisant le RB sous-jacent. Les algorithmes *classiques* utilisés sont : *Variable Elimination* (VE) (Dechter, 1999), *Shafer-Shenoy*

(SS) (Shenoy *et al.*, 1990) et *Variable Elimination* couplé avec *BayesBall* (VEBB) (Shachter, 1998). Ces résultats montrent la limitation d'employer un RB sous-jacent pour l'inférence dans les PRM, puisque l'ordinateur utilisé pour les tests ne pouvait pas représenter en mémoire de tels RB. De plus, SVE et SVED fournissent de bons résultats.

La figure 4b montre le comportement des algorithmes SVE et SVED en l'absence d'observation. Le système utilisé possède quarante salles avec cinquante ordinateurs et un nombre d'imprimantes variant de un à dix (pour un total de $6\,041$ à $6\,401$ nœuds). Les courbes montrent clairement le gain possible qu'offre une analyse en d-séparation. Toutefois, nous rappelons que l'exemple utilisé est un meilleur cas puisqu'il possède une topologie propice à l'analyse en d-séparation. Cependant, dans le pire cas SVED tendra vers SVE puisqu'il n'élaguera aucun attribut du système (la complexité de *BayesBall* étant polynomiale, le coût engendré par l'analyse en d-séparation devient rapidement négligeable en regard de celui de l'inférence).



(a) Avec six imprimantes par salle. (b) Avec huit imprimantes par salle.

Figure 5. *Performances des algorithmes SVE et SVED en la présence d'observations.*

La figure 5a montre l'impact des observations sur les algorithmes SVE et SVED dans un système avec six imprimantes, cinquante ordinateurs et quarante salles. Le nombre d'attributs observés dans chaque salle varie de zéro à cinquante (une observation par ordinateur). Pour l'exemple utilisé, ce type de configuration d'observation est un pire cas et nous pouvons constater que les performances de SVED se dégradent pour tendre, puis dépasser celles de SVE. La figure 5b reprend l'expérience de la figure 5a, mais nous avons augmenté le nombre d'imprimantes de six à huit. Ceci nous montre que le coût de l'analyse en d-séparation devient rapidement négligeable en regard du coût de l'inférence.

7. Conclusion

Nous avons proposé une adaptation de l'algorithme *BayesBall* pour SVE. Cette modification permet d'obtenir un gain de performance non négligeable pour le problème de l'inférence dans les PRM. De plus, ces travaux offrent des résultats expérimentaux de l'inférence dans les PRM, résultats absents de l'état de l'art actuel. Ces

résultats confirment l'utilité de développer de nouvelles méthodes d'inférences dans les PRM et montre qu'il reste encore de nombreuses possibilités de recherche sur cette problématique. En effet, nous n'avons qu'effleuré les nouveautés apportées par les PRM, tel que l'héritage de classe ou de type, l'incertitude structurelle ou encore l'incertitude existentielle. Ainsi, *Variable Elimination* n'est plus le seul algorithme d'inférence adapté aux PRM.

Remerciements

Nos remerciements s'adressent à la DGA pour le financement de ma thèse et ainsi qu'à l'ensemble des partenaires du projet ANR SKOOB : Bayesia, le CHU de Nancy, le CRAN, EDF R&D, l'ERPI, l'INERIS et la SOREDAB.

8. Bibliographie

aGrUM, « aGrUM : a GRaphical Universal Model », 2008. http ://agrum.lip6.fr.

Bangsö O., Wuillemin P.-H., « Top-Down Construction and Repetitive Structures Representation in Bayesian Networks », *Proceedings of the 13th Florida Artificial Intelligence Research Society Conference*, p. 282, 2000.

Dechter R., « Bucket Elimination : A Unifying Framework for Probabilistic Inference », *in* M. I. Jordan (ed.), *Learning in graphical models*, MIT Press, 1999.

Domingos P., Richardson M., « Markov Logic : A Unifying Framework for Statistical Relational Learning », *in* L. Gettor, B. Taskar (eds), *Introduction to Statistical Relational Learning*, MIT press, p. 339-372, 2007.

Getoor L., Taskar B., *Introduction to Statistical Relational Learning (Adaptive Computation and Machine Learning)*, The MIT Press, 2007.

Jaeger M., « Relational Bayesian Networks », *UAI-97*, Morgan Kaufmann, p. 266-273, 1997.

Kerstring K., De Raedt L., « Bayesian Logic Programs », *Artificial Intelligence*, 2000.

Koller D., Pfeffer A., « Object-Oriented Bayesian Networks », *Proceedings of the 13th Annual Conference on Uncertainty in AI (UAI)*, p. 302-313, 1997.

Laskey K. B., « MEBN : A Language for First-Order Bayesian Knowledge Bases », *Artificial Intelligence*, vol. 172, p. 140-178, 2008.

Pearl J., *Probabilistic Reasoning in Intelligent Systems*, Morgan Kaufmann, San Mateo, CA, 1988.

Pfeffer A. J., Probabilistic Reasoning for Complex Systems, PhD thesis, Stanford University, January, 2000.

Shachter R., « Bayes-Ball : The rational pastime », *Proceedings of the 14th Annual Conference on Uncertainty in AI (UAI)*, Morgan Kaufmann, p. 480-487, 1998.

Shenoy P., Shafer G., « Axioms for probability and belief-function propagation », *in* G. Shafer, J. Pearl (eds), *Readings in Uncertain Reasoning*, Morgan Kaufmann, 1990.

SKOOB, « The SKOOB ANR project », 2007. http ://skoob.lip6.fr.